

RAPPORT DE *Stochastic Optimization and Automatic Differentiation for Machine Learning*

Prédiction du lieu d'origine d'une musique

Sommaire

1	Introduction	1
2	Estimation des coordonnées du lieu d'origine d'une musique	1
3	Estimation de <i>heatmap</i>	3
4	Conclusion	5
5	Annexes	5

1 Introduction

J'ai dans ce projet implémenté des réseaux de neurones à deux couches afin d'apprendre à déterminer le lieu d'origine d'une musique à partir de caractéristiques données. Pour cet exercice, j'ai dû utiliser les données de *Geographical Origin of Music*¹. Elles concernent 1059 "musiques du monde / traditionnelles / ethniques" provenant de 33 pays ou régions différents (pas de musique occidentale dont l'influence est trop répandue pour pouvoir déterminer de lieu d'origine précis). Nous disposons pour chacune de ces musiques de caractéristiques représentées par 116 variables ainsi que 2 variables indiquant les coordonnées géographiques de la région d'origine de la musique (latitude et longitude). Cependant, j'ai remarqué que quelques variables sont identiques alors j'ai supprimé les doublons ce qui nous ramène à 72 variables explicatives.

J'ai dans un premier temps construit un réseau de neurone afin de déterminer les coordonnées géographiques à partir des caractéristique d'une musique. Par la suite, j'ai remplacé les données de coordonnées géographiques par des heatmaps créées par les soins de M. Cuturi permettant d'indiquer les zones probables de provenance des musiques plutôt que d'indiquer un point précis avec des coordonnées géographiques. J'ai alors construit un deuxième réseau de neurone pour cette fois déterminer non pas les coordonnées mais la heatmap à partir des caractéristiques d'une musique.

2 Estimation des coordonnées du lieu d'origine d'une musique

J'ai d'abord cherché à faire un réseau de neurone avec deux couches afin de pouvoir estimer les coordonnées géographiques du lieu d'origine d'une musique à partir de ses caractéristiques. Comme on dispose de 72 variables de caractéristiques, on a un réseau de cette forme :

$$f(x) = \tanh(A_1\sigma(A_2x)) \quad (1)$$

où $x \in \mathbb{R}^{72}$ est le vecteur de caractéristiques d'une musique, A_2 est une matrice de dimension $p \times 72$, σ une fonction d'activation non linéaire et A_1 est une matrice de dimension $2 \times p$ afin de revenir à un vecteur de même dimension que les coordonnées. Enfin, la fonction \tanh transforme le résultat et le borne sur $[-1, 1]$. Il reste ensuite à dilater par une transformation linéaire cet intervalle afin de revenir dans l'échelle des coordonnées. Pour la fonction σ , j'ai choisi d'utiliser la fonction \tanh . Nous avons alors à choisir le nombre p qui détermine la taille de la couche intermédiaire du réseau.

Par ailleurs, il faut aussi déterminer une fonction de perte (ou une fonction objectif) afin d'avoir une fonction à optimiser pour le réseau de neurones. On prend naturellement la distance entre le point prédit et le vrai point comme fonction de perte. Cependant, comme les points sont données par coordonnées géographiques, on utilise la *distance du grand cercle*² définie par :

$$D = 2R \arcsin \sqrt{\sin^2 \left(\frac{\delta' - \delta}{2} \right) + \cos \delta \cos \delta' \sin^2 \left(\frac{\lambda' - \lambda}{2} \right)} \quad (2)$$

avec R , le rayon de la Terre, δ la latitude et λ la longitude.

À cette distance, nous ajoutons une pénalisation L_2 des paramètres du réseau de neurone ce qui accroît la perte en fonction de la norme des paramètres. afin d'éviter le surapprentissage (*overfit*). Le poids de la pénalisation est un hyperparamètre de plus à choisir.

1. <https://archive.ics.uci.edu/ml/datasets/Geographical+Origin+of+Music>
2. https://fr.wikipedia.org/wiki/Distance_du_grand_cercle

On peut noter qu'on peut se passer dans l'expression de D du facteur $2R$ devant puisqu'en divisant D ainsi que le paramètre de pénalisation par $2R$ on obtient une transformation croissante (homothétie positive) de la fonction de perte, ce qui ne change rien à notre problème puisqu'on cherche à minimiser cette fonction.

J'ai procédé par validation croisée (*cross-validation*) afin de déterminer les hyperparamètres marchant le mieux pour notre problème ; j'ai fait varier :

- Le paramètre de pénalisation L_2 entre 0,001 et 100
- p , le nombre de neurones dans la couche intermédiaire entre 10 et 200
- Le pas d'apprentissage entre 0,001 et 0,05 (par "pas d'apprentissage", j'entends le facteur par lequel on multiplie le vecteur de descente avant la mise à jour des paramètres à chaque itération)

Certains hyperparamètres sont restés constant, comme le nombre d'itérations fixé à 200, ou encore "l'inertie" : à chaque calcul de gradient de la fonction de perte, on garde 90% du vecteur précédent de descente et on ajoute 10% du dernier gradient calculé.

La base de données a été séparée de manière aléatoire en deux bases : une d'entraînement constituée de 80% des observations ainsi qu'une base de test, constituée des 20% restants. Chaque réseau de neurones est entraîné sur la base d'entraînement et on conserve celui qui fait la plus petite erreur (la distance D) en moyenne sur la base de test. Les meilleurs résultats sont obtenus pour un paramètre de pénalisation égal à 1, 10 neurones intermédiaires ainsi qu'un pas de 0,001. L'erreur moyenne obtenue est alors de 0,25 en omettant le facteur $2R$, ce qui nous ramène à 3165 km sachant que la Terre a un rayon moyen de 6371 km.

L'histogramme en figure 1 donne la répartition des erreurs : la distribution est asymétrique, la médiane est à 2650 et les erreurs sont parfois très grandes, pouvant dépasser les 10000 km

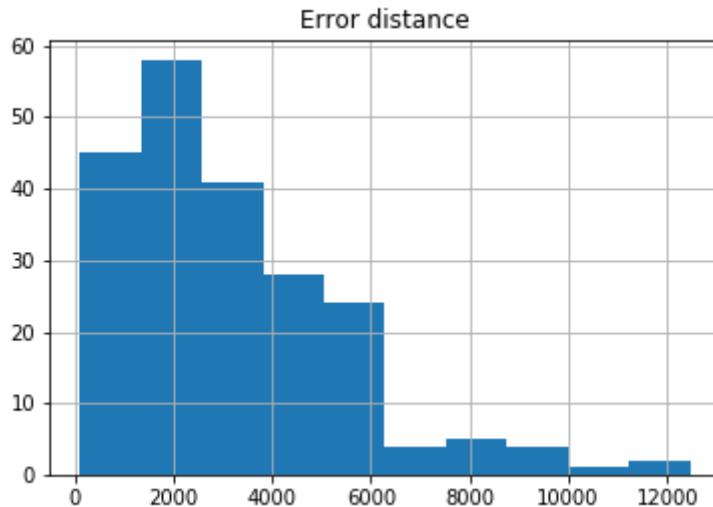


FIGURE 1 – Histogramme des erreurs

J'ai représenté sur une carte (figure 2) les lieux d'origines de quelques musiques de la base de test ainsi que les prédictions associées. Le point correspondant aux coordonnées indiquées dans les données est représenté par une icône de nuage. La prédition associée à la même musique est représenté par la même couleur mais avec une icône de "i". Par soucis de lisibilité, je n'ai sélectionné (de manière aléatoire) qu'une dizaine de musiques. On observe que les prédictions sont souvent pas très éloignées des vraies coordonnées. Cependant, l'écart est parfois assez grand, avec par exemple une musique provenant du nord de l'Europe de l'est (en rose) a été prédit dans le Moyen-Orient.



FIGURE 2 – Carte représentant la provenance de quelques musiques de la base de test (icône de nuage) ainsi que la prédiction associée (icône de "i", même couleur)

Le résultat n'est pas extrêmement précis mais il est quand même plus informatif qu'une prédiction au hasard, sachant que la circonference de la terre est de $40000\ km$. De plus une approximation de $3000\ km$ semble pas si mal sachant qu'on cherche le lieu d'origine d'une musique (ce qui peut être imprécis) à partir de ses caractéristiques. Par ailleurs, le jeu de données a été construit en attribuant les coordonnées de la capitale du pays de résidence de l'artiste (ou la ville principale de la région). Il est donc normal qu'on soit capable que de donner qu'approximativement le lieu d'origine de la musique, et on préférerait donner des zones probables d'où pourrait provenir la musique plutôt que de donner un point précis. C'est pour cela que dans la suite, nous ne localisons plus les musiques par des coordonnées géographiques mais par une *heatmap* qui indique les zones probables de provenance de la musique.

3 Estimation de *heatmap*

On cherche donc maintenant à trouver les zones les plus probables d'où pourraient provenir les musiques. On fournit donc pour chaque musique non pas deux coordonnées mais une *heatmap* de taille 20×20 . On découpe donc la carte en un quadrillage de 20×20 cases. On associe à chaque case un nombre indiquant la probabilité que le lieu d'origine de la musique se trouve dans cette case.

On construit un nouveau réseau de neurone pour pouvoir apprendre à construire cette *heatmap* à partir des caractéristiques de la musique. Le modèle est assez similaire à ce qu'on avait précédemment :

$$f(x) = s - \max(A_1 \sigma(A_2 x)) \quad (3)$$

avec pour différence que A_1 est de dimension $400 \times p$ puisqu'on a besoin d'un nombre pour chacune des 400 cases. De plus on a remplacé la dernière fonction d'activation par $s - \max$ le *softmax* défini par :

$$s - \max(X)_i = \frac{e^{x_i}}{\sum_k e^{x_k}} \quad (4)$$

Cette fonction nous assure que nous disposons à la fin d'une heatmap qui indique bien des probabilités (chaque coefficient est alors positif et leur somme vaut 1).

Par ailleurs nous changeons également la fonction de perte en remplaçant la distance du grand cercle par la divergence de Kullback-Leibler, qui permet de calculer une divergence entre la distribution de probabilité prédite et la distribution de probabilité figurant dans la vraie *heatmap*. Elle est définie de la manière suivante :

$$D_{KL}(P||Q) = \sum_i P(i) \log \left(\frac{P(i)}{Q(i)} \right) \quad (5)$$

avec i parcourant toutes les cases de la *heatmap*, $P(i)$ la probabilité associée à la case i dans la vraie *heatmap* et $Q(i)$ la probabilité associée à cette même case dans la *heatmap* prédite. J'ai rencontré une difficulté technique avec cette divergence puisque les calculs devaient être faits par le module *numpy* d'*autograd* pour pouvoir ensuite calculer automatiquement le gradient. Cependant, en écrivant la formule telle quelle, on rencontre des problèmes pour les cases où $P(i)$ vaut 0 car le logarithme de 0 n'est pas défini. Ceci n'est pas un problème théorique car on ne somme que sur les cases où $P(i)$ est non nul. J'ai réussi à contourner le problème en réécrivant la divergence de Kullback-Leibler de la manière suivante :

$$D_{KL}(P||Q) = \sum_i \log \left(\left(\frac{P(i)}{Q(i)} \right)^{P(i)} \right) \quad (6)$$

Bien que les deux expressions sont mathématiquement identiques, cette réécriture permet de résoudre ce problème computationnel. En effet, lorsque $P(i)$ vaut 0, l'intérieur du logarithme est évalué à 1 par *numpy* et quand on applique enfin le logarithme, on retombe bien sur 0.

Comme précédemment, on ajoute une pénalité L_2 et on procède par validation-croisée pour choisir les hyperparamètres. Cette fois-ci, j'ai fait varier :

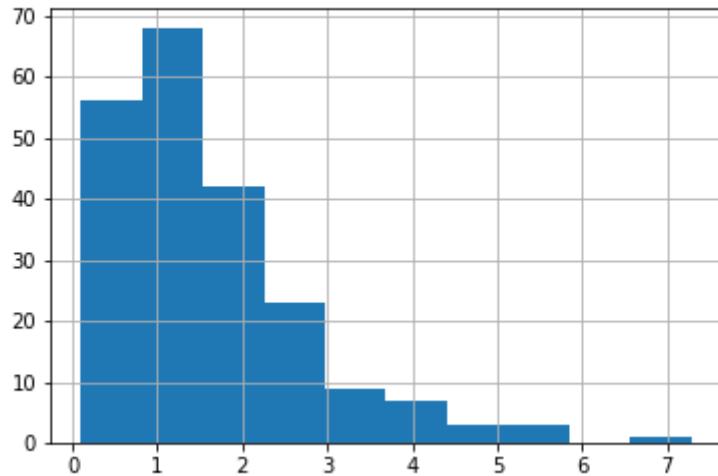
- Le paramètre de pénalisation L_2 entre 0 et 100 ; comme il y a plus de paramètres, je me suis dit qu'il faudrait sûrement une pénalisation moins forte que précédemment, voire éventuellement nulle donc j'ai commencé à 0
- p , le nombre de neurones dans la couche intermédiaire entre 10 et 500 ; comme la dimension de la sortie est beaucoup plus grande je me suis dit qu'il faudrait peut-être plus de neurones intermédiaires, je voulais au début tester jusqu'à 2000 mais l'algorithme mettait beaucoup trop de temps à tourner.
- Le pas d'apprentissage entre 0,001 et 0,05 (inchangé)

La même séparation entraînement/test a été faite. On obtient finalement une erreur de test minimale pour un paramètre de pénalisation L_2 égal à 0,001, un nombre de neurones intermédiaires de 100 et un pas de 0,05. On obtient alors une divergence de Kullback-Leibler moyenne de 1,59. Comme la convergence de l'erreur semblait plus lente que dans le modèle précédent (l'erreur semblait descendre encore un peu), j'ai continué l'optimisation sur 1000 itérations de plus pour les hyperparamètres déterminés ci-dessus. L'erreur moyenne de test est alors descendue à 1,56.

On retrouve encore une fois une distribution asymétrique des erreurs avec de temps en temps de grosses erreurs et une médiane à 1,33 (figure 3).

J'ai représenté en annexe les *heatmaps* réelles et prédites de quelques musiques de la base de test choisies aléatoirement. La prédiction est parfois très bonne (figures 4 et 5). La prédiction est parfois un peu imprécise (figures 7 et 8). Et il arrive aussi que la prédiction soit complètement fausse (figures 9 et 10).

Quand j'ai refait 1000 itérations pour l'entraînement du réseau de neurone, l'erreur sur la base d'entraînement est descendue de 1,18 à 0,70. Comme l'erreur sur la base d'entraînement est beaucoup plus faible, on peut suspecter un surapprentissage. J'ai alors entraîné un autre réseau de neurone avec cette fois-ci une pénalité L_2 avec un coefficient de 0,01 sur 1000 itérations. En effet, il est possible que les 200 itérations lors de la validation croisée étaient insuffisantes et que la

FIGURE 3 – Histogramme des erreurs des *heatmaps*

convergence n'avait pas eu le temps d'avoir lieu ce qui aurait pu induire une mauvaise identification des hyperparamètres optimaux. On obtient alors une erreur de test et d'entraînement similaires, respectivement à 2,49 et à 2,40 ce qui indiquerait qu'il n'y a pas de surapprentissage. Cependant, les erreurs sont plus élevées que précédemment donc le modèle précédent est préférable.

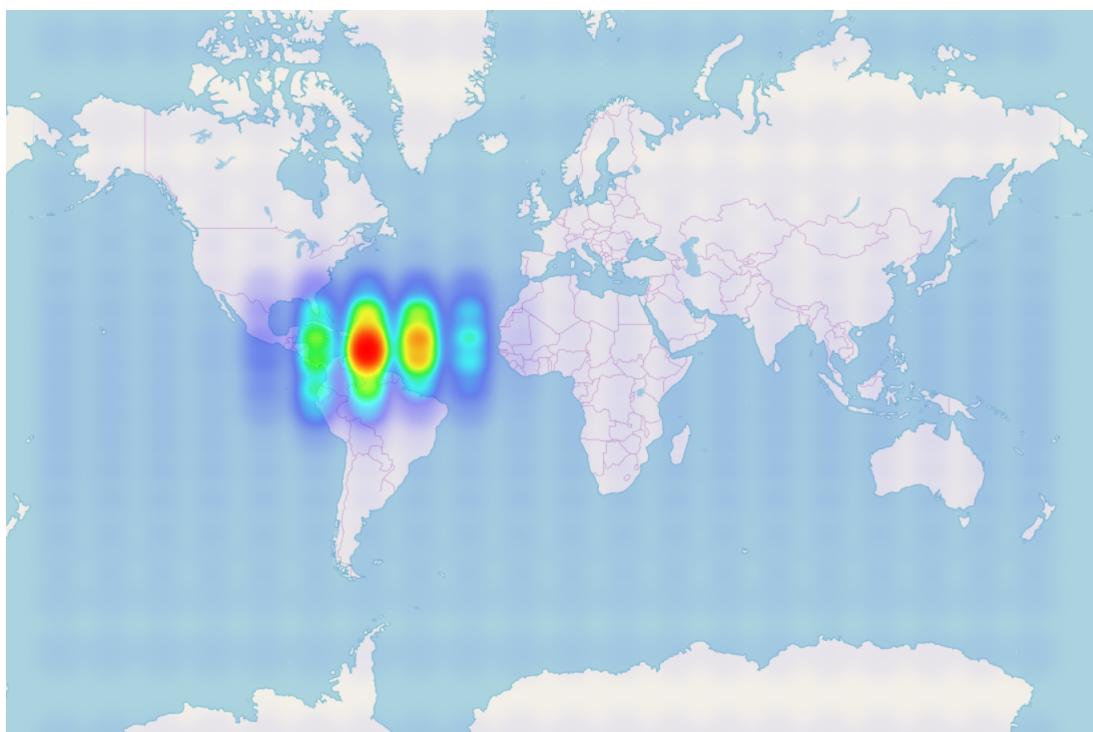
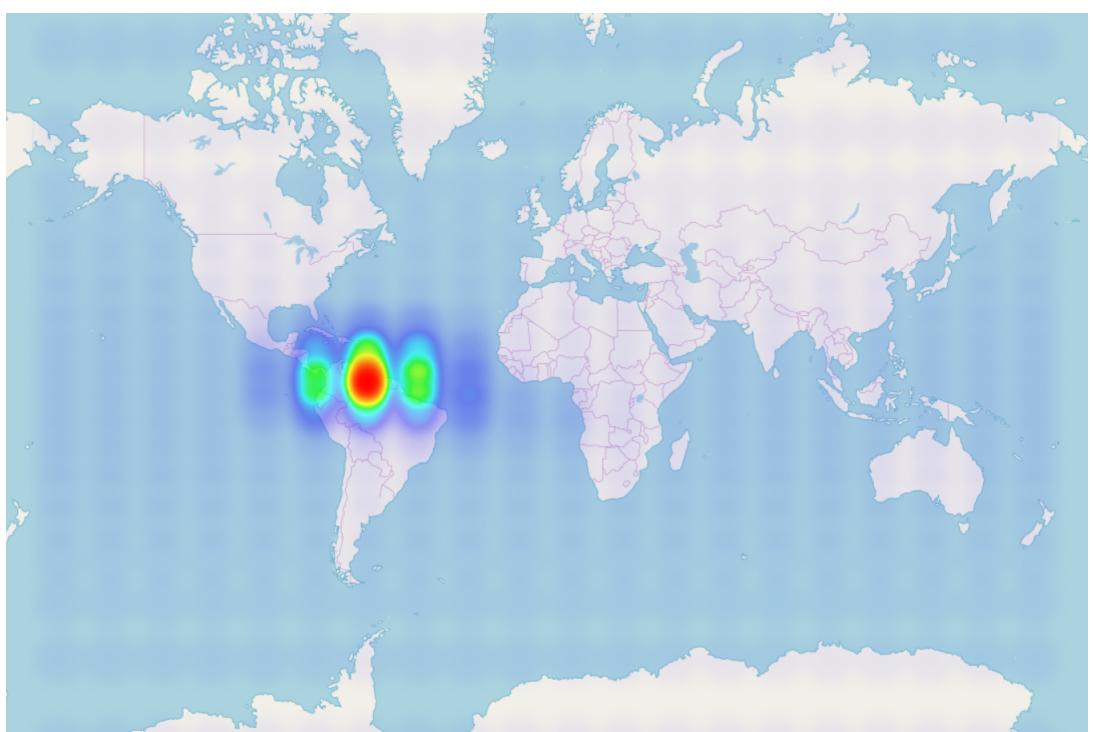
On observe d'ailleurs sur les *heatmaps* produites que les résultats sont moins précis et plus souvent complètement faux (figure 6).

4 Conclusion

La différenciation automatique est pratique pour l'entraînement de réseau de neurone. Il faut cependant bien choisir quelques paramètres. Pour ce faire, la validation croisée semble la meilleure solution.

Concernant la prédiction du lieu d'origine des musiques, on observe d'abord que les réseaux de neurones ne permettent pas d'avoir une très grande précision sur le lieu d'origine d'une musique mais ils permettent toutefois d'avoir une estimation pas trop éloignée du vrai lieu. En passant aux *heatmaps*, on peut alors déterminer des zones probables et on observe que les réseaux de neurones permettent d'avoir une bonne approximation.

5 Annexes

FIGURE 4 – Vraie *Heatmap* 1FIGURE 5 – *Heatmap* prédictive 1 : bonne prédition

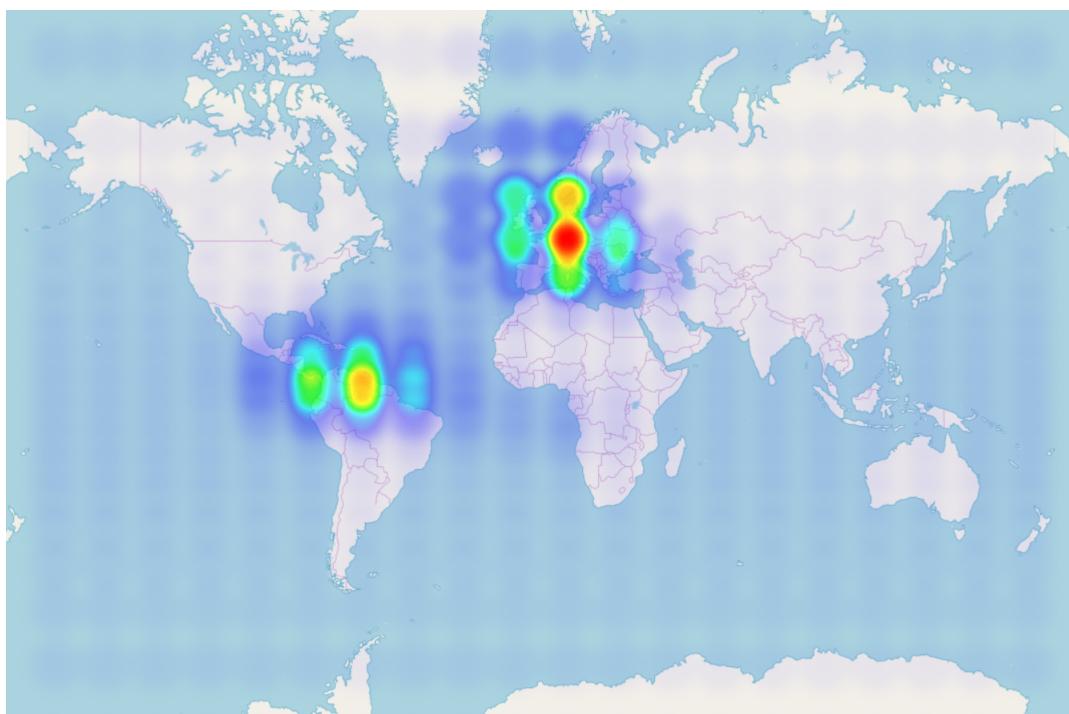


FIGURE 6 – *Heatmap* prédite 1 avec une plus grosse pénalité L_2 : prédiction moins précise

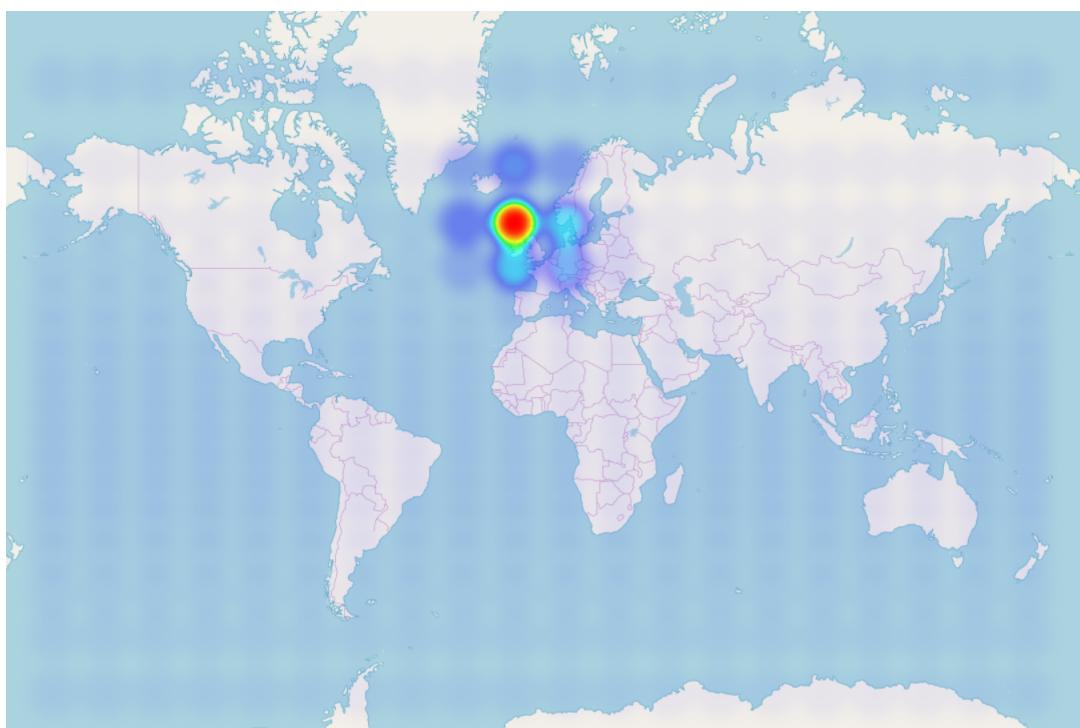


FIGURE 7 – Vraie *Heatmap* 2

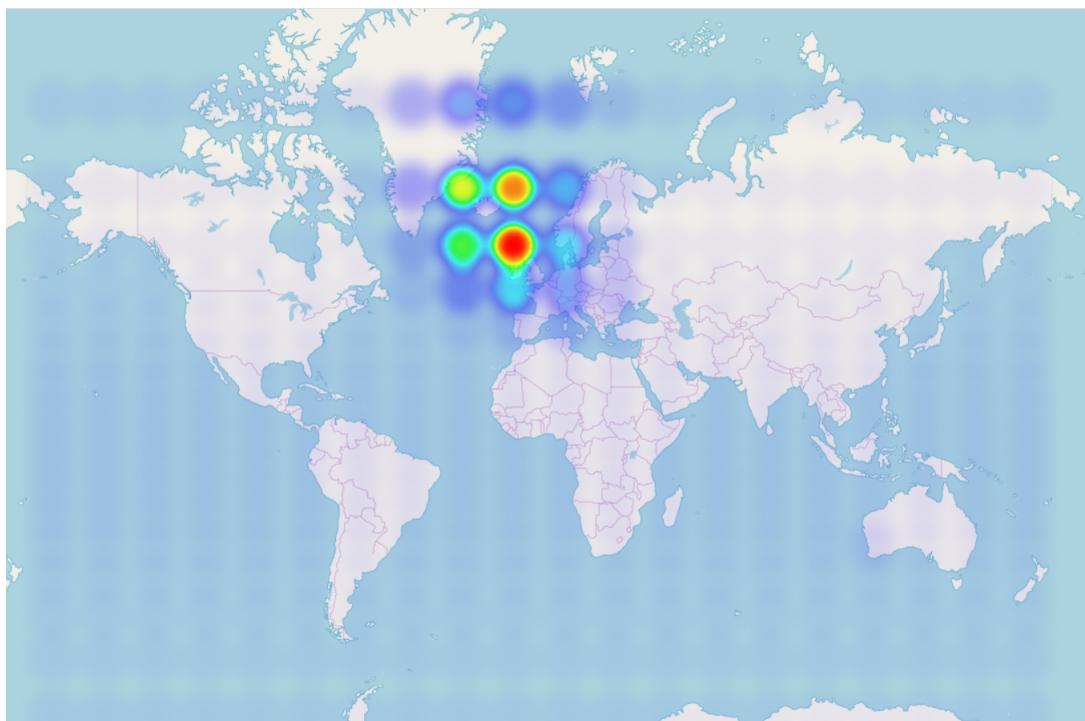


FIGURE 8 – *Heatmap* prédite 2 : prédiction un peu imprécise

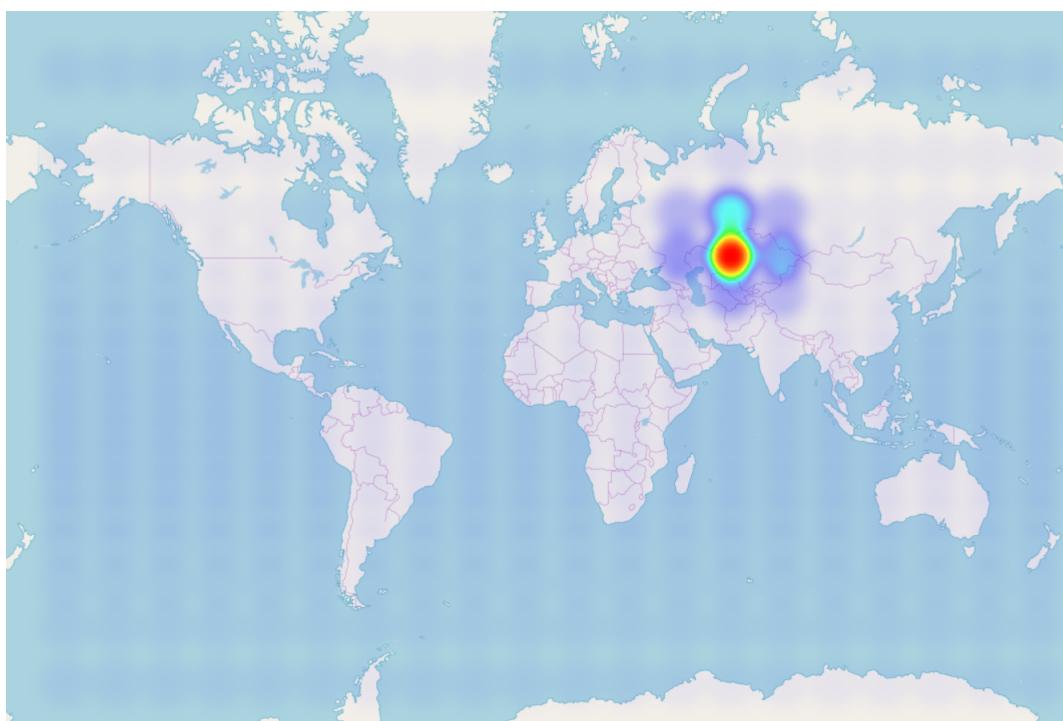


FIGURE 9 – Vraie *Heatmap* 3

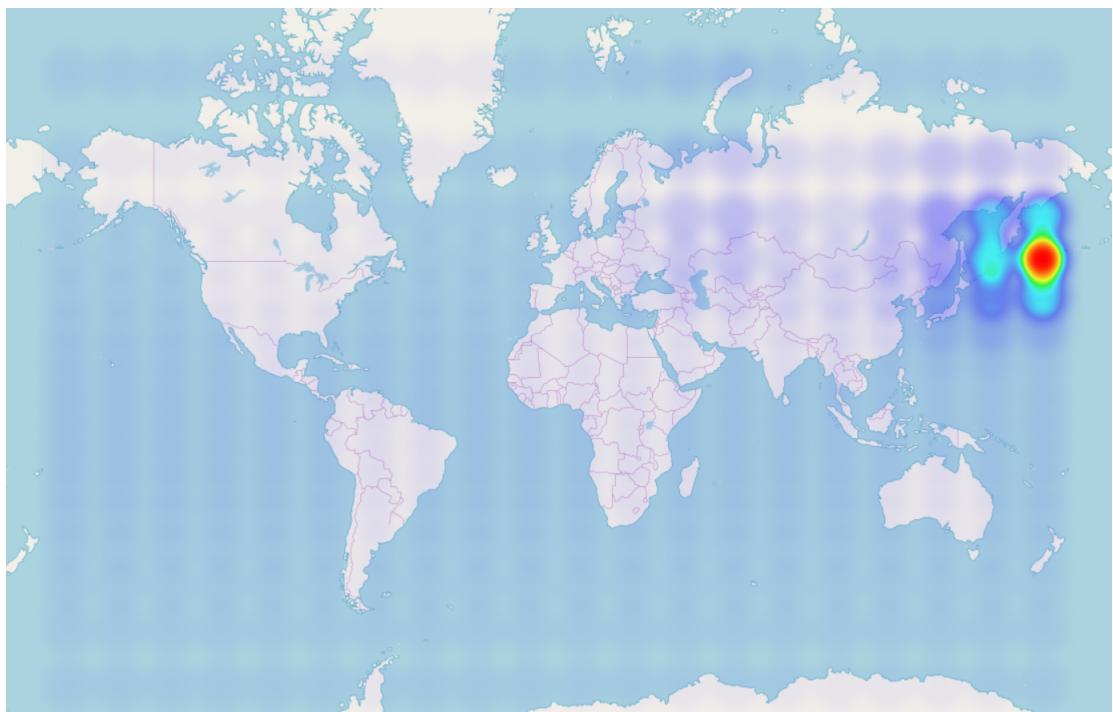


FIGURE 10 – *Heatmap* prédite 3 : prédiction complètement erronée